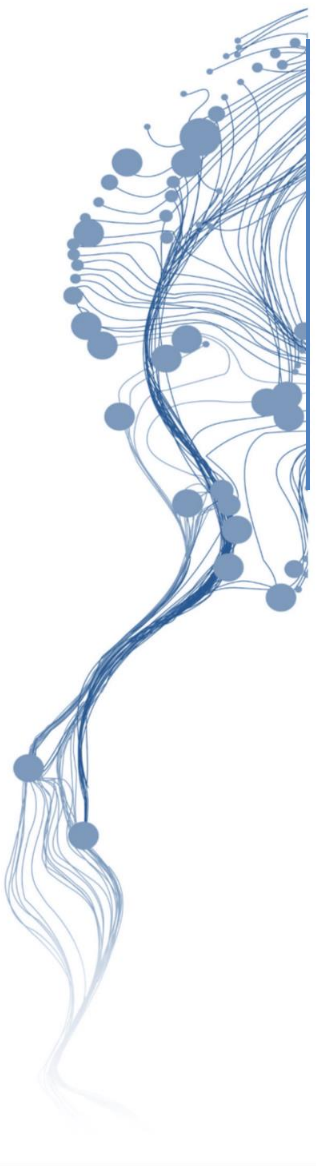


City of Amsterdam – ITC Twente

Pole-like objects part-segmentation

Internship Project

Sara Yousefimashhoor
Dec 2021 – Mar 2022

Contents

1. Introduction

1.1. Background

Preserving and managing urban assets is one of the main features of any sustainable city. Urban asset management requires a high-quality inventory representing the existing assets' location, type, and condition. This inventory would highlight the priority action areas and help reduce maintenance costs for the community.

With the advancement of remote sensing technologies, LiDAR data is becoming a more popular and affordable source to extract and identify assets in outdoor scenes as they provide accurate geometry information and are independent of lighting conditions (Guo et al., 2020; Karlsson, 2014; Li et al., 2021). City of Amsterdam, one of the pioneering municipalities implementing smart solutions for urban management, has started the point cloud project to extract relevant information from the LiDAR data automatically.

1.2. Problem and Objectives

This internship project is defined under the point cloud project in the Innovatieteam (CTO) of the Amsterdam municipality. The particular focus of this internship project is to classify isolated pole-like objects[1] (PLO) found in urban scenes. This classification[2] can enrich the topographical map with more details about the types of components attached to every pole. This detail would highly benefit the facility management department.

The main challenge of this project is the classification of complex instances of poles with multiple attachments. Many attachments can be found on pole-like objects in urban scenes, such as directional signs, traffic lights, street signs, street lights, horizontal poles, etc. Assigning one descriptive type to the complex poles is impossible. Therefore, to include and describe all possible pole instances, the main goal of this project is to decompose individual poles to their constituent components (Figure 1) and determine what type of extensions are attached to the carrier of the pole.

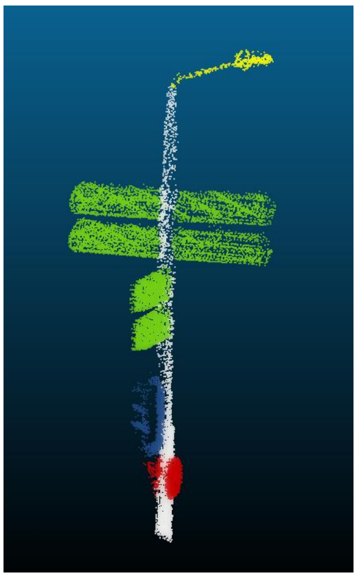


Figure 1 - Expected Result

So far, the point cloud group of the AI department has extracted poles from a point cloud provided by Cyclomedia using a semantic segmentation deep learning algorithm (RandLA-Net). Also, they have generated a dataset of 262 labeled pole-like objects, which is the main data for any machine-learning task in this project.

1 - By "pole-like objects" we mean cylindrical objects with upward direction along the road and their extensions.

[2] *By "classification" we mean to determine whether a pole is a light pole, or a traffic sign, or a traffic light, or a bollards, or a street sign, etc.*

1.3. Report structure

This report is structured within four main sections: In Section(2), the previous related scientific works are being reviewed. Section(3) presents a brief description of the selected methods for decomposing poles. Section (4) is the core of this report, including the experiments and the results of both approaches. Finally, Section(5) evaluates the methods and includes the project conclusion and suggestions for improving the outcomes.

2. The Literature Review

2.1. The common workflow

There is rich scientific literature focusing on pole recognition (detection and classification). The majority of scholars have used handcrafted features, prior knowledge, or predefined models to classify poles. The urban poles' standard shape, height, and size have led these methods to work properly for simple instances of carriers with one attachment. The common workflow of the pole recognition process from a point cloud includes three steps:

2.1.1. data preprocessing

As an unstructured, heavy, and noisy data type, point cloud requires some modification before processing to make the problem easier to solve. The point cloud is preprocessed (smoothing and denoising) to remove the outliers and unwanted objects (ground filtering, buildings, trees, dynamic objects removal). Some preprocessing strategies like partitioning the point cloud and down-sampling help reduce computational efforts. Some other techniques like clustering off-ground points, voxelization, and 2D projection would give structure to the data.

2.1.2. Pole detection

pole extraction is one of the main concerns in the pole recognition pipeline. In this project, poles are already extracted using semantic scene segmentation deep learning algorithm. However, the previous scientific works would identify the poles using cross-section analysis, scan-line pattern, cylinder/line fitting in 3D, circular/arc fitting in 2D, calculating eigenvalues and vectors (and determining based on the linearity), setting height thresholds, and vertical and horizontal region growing.

2.1.3. Pole classification

After extracting the pole-like objects, they are classified into different subcategories (street light, street sign, directional sign, traffic light, etc.). The majority of the published papers are focused on poles with one attachment. However, many poles contain multiple attachments. The concept of pole decomposition(Li et al., 2018) is developed to segment the pole-like object into a carrier and its attachments. Usually, the points belonging to individual attachments are grouped using connected component analysis after removing the main carrier. It is beneficial to separate carriers from extensions before extracting feature values because pole-like objects, as a whole, have similar feature values that do not help distinguish them further(Fukano and Masuda, 2015). Analyzing the attachments' geometric and radiometric features helps identify the individual attached items.

- **Geometric features** such as the size of the bounding rectangle (2D projection) or bounding box (3D), area (2D) and volume (3D) of the BBox, edges, axes, diagonals ratio, the pattern of points' scattering (linear, planar, volumetric), mean/median height of the feature, number of points, point density, and roughness are being widely used for the classification task. Also, template matching and calculating the cross-correlation and the RMSE is another approach for detecting an attachment of interest.

- **Radiometric features** such as the number of returning pulses (to distinguish trees), intensity (for markings and signs), and color information are used to differentiate among the attachments.

2.2. Typology of approaches

In general, methods to classify poles can be categorized into the following groups:

2.2.1. Model-based / Template-based

In this approach, a predefined model is generated for each object instance using a priori knowledge of the object properties to be matched and fitted to the point cloud for pole-like object classification.

2.2.2. Knowledge-based / Rule-based

Some rules are considered based on prior knowledge of the feature properties (e.g., spatial, geometric, or radiometric characteristics). Later, a machine-learning classifier can be used to identify subcategories of the pole-like objects based on these handcrafted features.

2.2.3. Deep learning-based

With the advancement of deep learning algorithms and the availability of 3D point cloud datasets, there is an opportunity to use fully automatic end-to-end pole-like object identification without generating models, creating rules, or handcrafting features.

Pole-like objects are usually identified as a byproduct of scene semantic segmentation tasks in many deep learning algorithms. However, further classification of poles to gain information about their attachments has not been investigated much.

In a recent paper, a DNN-based[3] method was proposed for the first time, specifically for pole-like objects' recognition (detection and classification)(Plachetka et al., 2021). Unfortunately, the authors shared neither the code nor their labeled dataset with the public. Also, in this work, poles are considered to have only one attachment. However, in reality, each pole carrier might contain several attachments, and various combinations of attachments make it almost impossible to solve this issue as a classification problem.

The most recent approaches to handling the complex pole instances using machine learning decompose the poles to their constituent parts and classify them individually. The same approach can be adopted in the deep learning domain as the part segmentation task.

In the part-segmentation task, a point cloud of an individual instance of an object class would be segmented to its constituent part, and a part label would be assigned to each point(Bello et al., 2020). Since the pipeline of a part-segmentation task is similar to semantic segmentation, semantic segmentation networks can be trained for part segmentation tasks(He et al., 2021). We are implementing the KPConv algorithm to decompose the poles for this internship project.

[3]- Deep Neural Network

3. Tested Methods

A number of criteria are considered in choosing our approach for this project, such as the limited time budget (3 months), the proficiency and the past experiences of the intern, the availability of the method to the public, and the method performance.

In this internship project, we aim to test two different approaches, one from the machine-learning domain with handcrafted features (CANUPO) and the other one by levering from the deep learning domain (KPConv).

3.1. CANUPO algorithm

CANUPO algorithm has a plugin in the CloudCompare software. This algorithm separates two classes by comparing their dimensionality on different scales. Dimensionality characterizes the arrangements of points within a sphere (Figure 2) with a certain radius, and it determines whether the organization of points is 1D (linear), 2D (planar), or 3D (volumetric) (Brodu and Lague, 2012). Exploring the dimensionality of a local set of points in multiple scales has resulted in the better separation and higher spatial resolution in the classification results (Figure 3). This method is developed to do the classification task for complex natural environments where classes of interest have organic geometry. In this project, we apply the algorithm for separating predefined man-made components and expect to achieve higher accuracy.

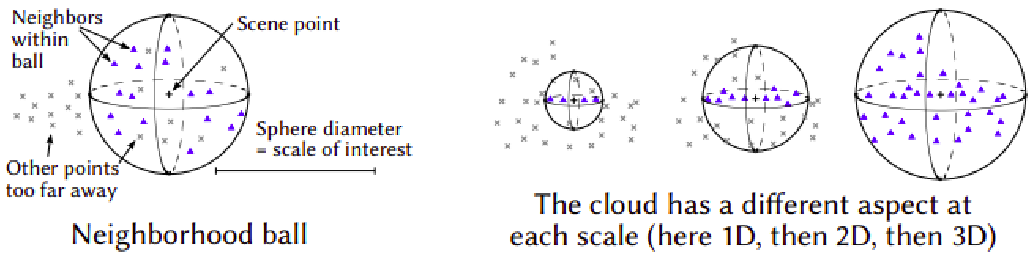

Neighbors within ball
Scene point
Sphere diameter = scale of interest
Other points too far away
Neighborhood ball
The cloud has a different aspect at each scale (here 1D, then 2D, then 3D)

Figure 2 - CANUPO Algorithm: the main concept | Neighborhood ball (Brodu and Lague, 2012, p.4)

This algorithm aims to achieve the signature by describing how each class's dimensionality changes on different scales and separating the classes based on this signature. This signature is called a multiscale local dimensionality feature (Brodu and Lague, 2012).

Scale is represented by the diameter of the ball centered around the point of interest (Figure 2). The dimensionality of the local neighborhood for a specific class is calculated on all of the given scales. However, they are considered with different weights based on their suitability for separating the two classes (Figure 3) (Brodu and Lague, 2012).

This algorithm typically uses a linear classifier like SVM to separate the classes in each scale. It is also using the Principal Component Analysis (PCA) to calculate the dimensionality of the local neighborhood on each scale (Brodu and Lague, 2012).

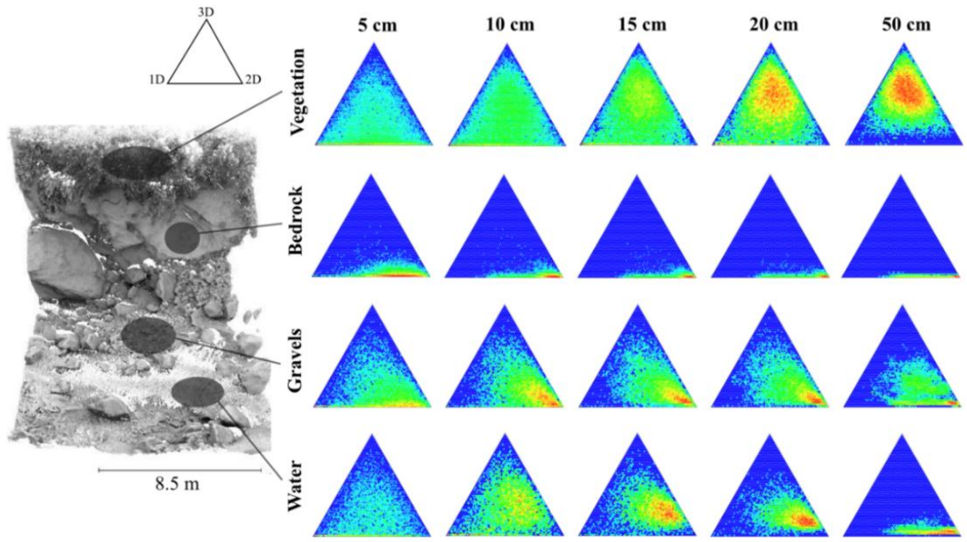


Figure 3- Degree of clustering around a given dimensionality in a multiscale and multi-class scenario (Brodu and Lague, 2012, p.11)

3.2. KPConv Algorithm

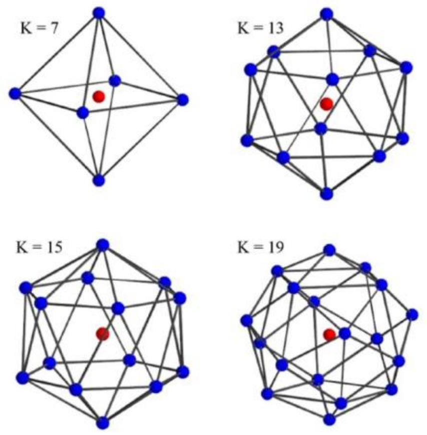


Figure 4 - Illustration of the kernel points in stable dispositions (Thomas et al., 2019)

Kernel Point Convolution (KPConv) is a point-based deep learning algorithm that consumes the points directly and applies the convolution to a spherical neighborhood using regularly arranged kernel points (Figure 4) (Thomas et al., 2019). Weights and the influence area for each kernel point are learned during the training phase (Figure 5), and all the kernel points will influence each point inside the neighborhood depending on its distance(Thomas et al., 2019). The convolution result will pass on to the next layer as input, and the process will be repeated (Figure 6).

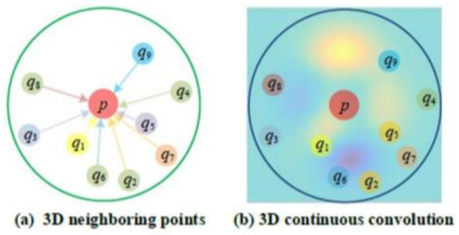


Figure 5 - Illustration of a continuous convolution for local neighbors of a point (p) and its neighboring points (qi) (Guo et al., 2020, p.5)

The KPConv algorithm is being used for several tasks such as semantic segmentation, part segmentation, and classification on benchmark datasets, and it is placed on the leaderboard for most of the datasets.

This project deals with decomposing a single isolated object with various part types and part numbers. Therefore the problem is close to the part segmentation task. The part segmentation task is for isolated individual objects with a known number of parts. The objective of this task is to divide the point cloud of the object into its constituent parts.

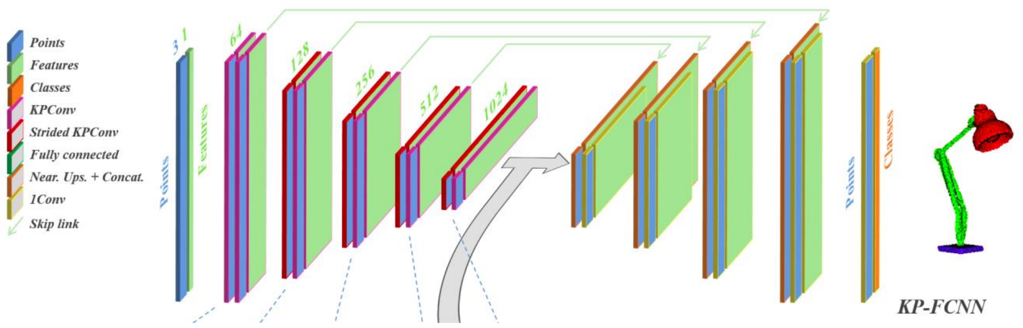


Figure 6 - Network Architecture for segmentation task (Thomas et al., 2019)

The KPConv part-segmentation task is originally tested on the ShapeNetPart[4] dataset. Some instances of the labeled objects existing in the ShapeNetPart dataset can be seen in Figure (7). In this project, we have modified the codes related to this benchmark dataset to do the part-segmentation task on our custom isolated poles.

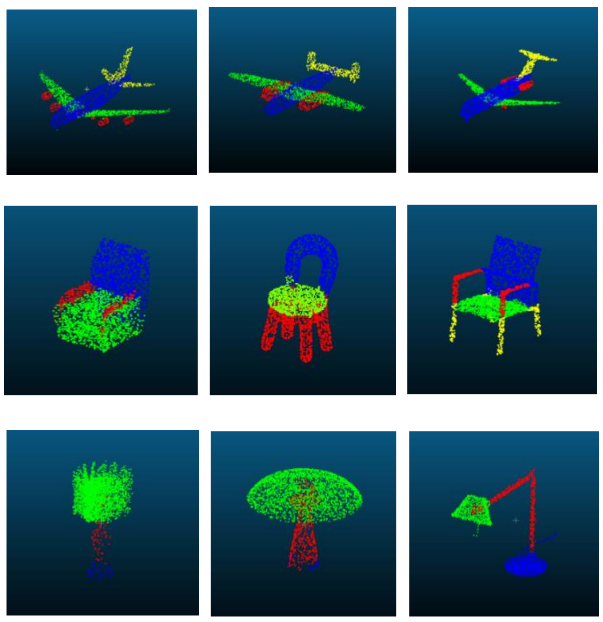


Figure 7 - ShapeNetPart Dataset Instances

4. Methodology

4.1. CANUPO Algorithm

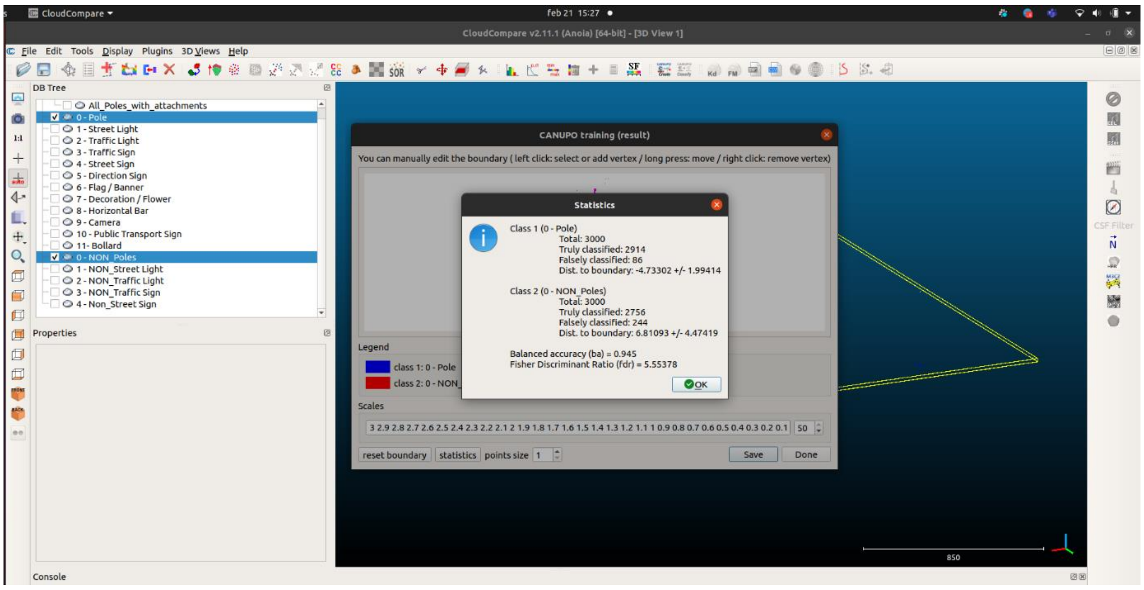
For the initial test with the CANUPO algorithm, the labeled pole dataset is partitioned into three sets of training, validation, and test (70% training (180 point clouds), 15% validation (41 point clouds), 15% test(41 point clouds)). The split method is the multilabel stratified shuffle split. Partitioning of data is done by considering the existence of all the attachment

[4] *This dataset was released in March 2019 by Stanford University and contains 15 object classes (airplane, bag, cap, car, chair, guitar, lamp, table, etc.).*

classes in each split. Since the CANUPO algorithm does not need the validation set, the validation and the test dataset are merged so that the same amount of training data makes both methods' (CANUPO and KPConv) results comparable.

To build the classifier, the CANUPO algorithm requires two layers of point clouds, each representing one class of interest. One hundred eighty instances of poles were imported and merged into a single layer as the training dataset. Then, twelve layers, each representing one type of pole part (e.g., pole, street light, traffic sign, traffic light, etc.), were extracted from the merged layer. At this point, the "one against the others" strategy was used for the binary classification. The pole class was considered layer one, and all other classes were merged to generate the "non-pole" class. The reason behind selecting the pole class is its linear dimensionality, which is unique enough to help separate it from almost all other classes. This is not the case for other classes because many can easily be confused due to similar dimensionalities. For example, street signs, traffic signs, and directional signs all have planar dimensionality and cannot be distinguished with this algorithm.

"Poles" and "non-pole" layers were fed into the CANUPO algorithm, generating a classifier. The input parameters were manually fine-tuned, and the best result is shown in figure (8).



You can manually edit the boundary (left click: select or add vertex / long press: move / right click: remove vertex)

```
Class 1 (0 - Pole)
        Total: 3000
        Truly classified: 2914
        Falsely classified: 86
        Dist. to boundary: -4.73302 +/- 1.99414

Class 2 (0 - NON_Poles)
        Total: 3000
        Truly classified: 2756
        Falsely classified: 244
        Dist. to boundary: 6.81093 +/- 4.47419

Balanced accuracy (ba) = 0.945
Fisher Discriminant Ratio (fdr) = 5.55378
```

Legend
🟦 class 1: 0 - Pole
🟥 class 2: 0 - NON_

Scales: 3 2.9 2.8 2.7 2.6 2.5 2.4 2.3 2.2 2.1 2 1.9 1.8 1.7 1.6 1.5 1.4 1.3 1.2 1.1 1 0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2 0.1 | 50

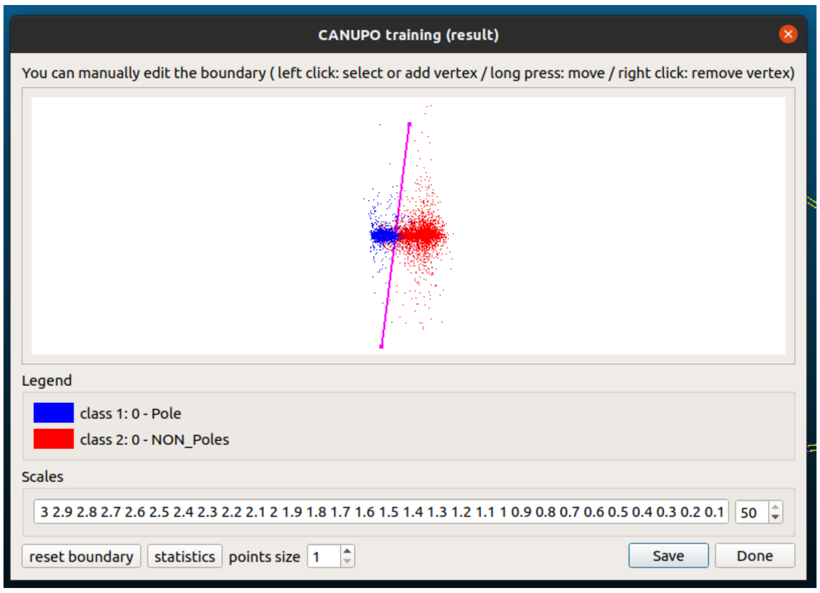
points size 1

Figure 8 - CANUPO algorithm training input and results

The max core points[5] parameter was set to 3000 for each class, and the scale changed from 0.1 m to 5m with the steps of 10cm. The balanced accuracy for the classification with the input parameters was 94.5%, and the FDR (Fisher Discriminant Ratio) of the two classes was 5.5. FDR measure represents the discriminative power (separability) of the classifier using a particular feature, and it is calculated by assigning a signed distance of each sample to the separation line[6] (Brodu and Lague, 2012). The visualization of the two classes and the classifier can be found in figure (9).

[5] The number of points to be considered for the computations from each class.

[6] $fdr = \frac{(\mu_2 - \mu_1)^2}{(v_1 + v_2)}$ where µ represents the mean and v represents the variance of each class


CANUPO training (result)

You can manually edit the boundary (left click: select or add vertex / long press: move / right click: remove vertex)

Legend

🟦 class 1: 0 - Pole

🟥 class 2: 0 - NON_Poles

Scales

3 2.9 2.8 2.7 2.6 2.5 2.4 2.3 2.2 2.1 2 1.9 1.8 1.7 1.6 1.5 1.4 1.3 1.2 1.1 1 0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2 0.1 | 50

Figure 9 - Illustration of the classifier in the CANUPO algorithm

4.2. KPConv Algorithm: Part-Segmentation Task

4.2.1. Data preparation:

Our custom dataset includes 262 .laz format point clouds containing coordinates, colors, and intensity values. To feed the custom data to the KPConv algorithm, the pole data was passed through the workflow shown in figure (10). The first step for data preparation was to split the data for training, validation, and test phases based on the existence of all the classes in all the splits[7]. After the split, 176 pole instances are dedicated to the training, while 43 pole instances are dedicated to each validation and test split.
The next step is to read the point clouds and normalize x,y,z, RGB, and intensity values to make learning the task easier for the model. For x,y,z normalization, the center of each instance is moved to the (0,0,0) coordinate. The 16bit RGB and intensity values are scaled between 0 and 1.
Some modifications like shifting the label values, renaming the files, and saving as .ply format are specific to the model to minimize the change in the code lines.

[7] *stratified split using Multilabel Stratified Shuffle Split functionality from iterstrat.ml _stratifiers library*

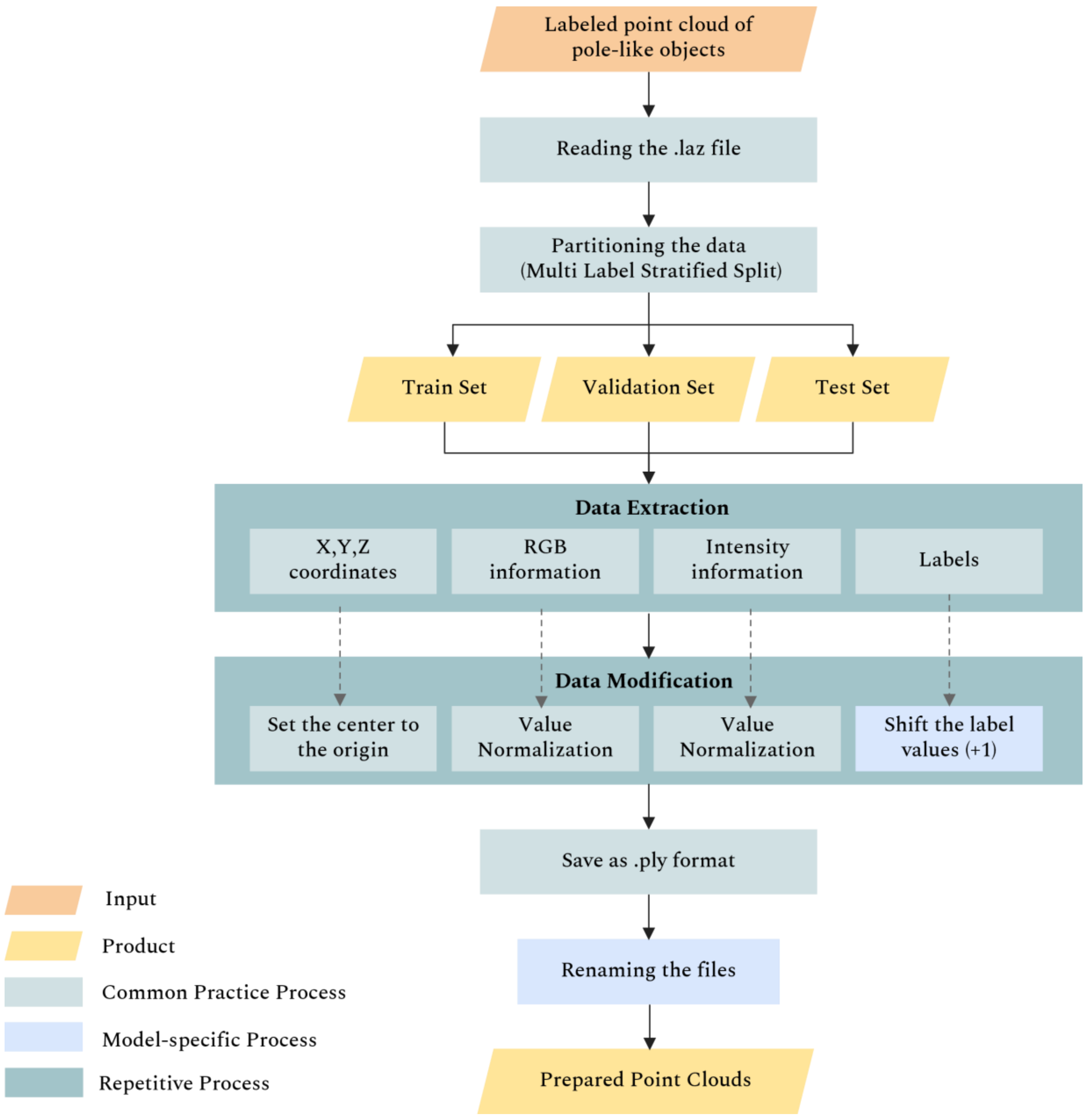


Figure 10 - The workflow of Data Preparation for a custom data to be fed to the KPConv algorithm for Part-Segmentation task

After the data preparation, the training data split is fed to the KPConv algorithm. The validation split is used to test the training every 50 steps. The trained model is being tested on the unseen test split, and the process was repeated for all the experiments.

5. Results

5.1. CANUPO Algorithm

As it can be seen in figure (11), the poles can be recognized from all other classes except the street lights and horizontal bars, which is understandable as these classes also have a linear dimensionality. The classifier has high confidence (Figure 12) about its prediction for most points except for the points belonging to the joints, bollards (short poles), and some edges of planar objects. However, the classifier has made a correct prediction about the mentioned positions. The high confidence is due to the high separability of the pole class from all other classes. The uncertainty about the intersections is predictable because two linear objects create a volumetric geometry when they intersect one another.

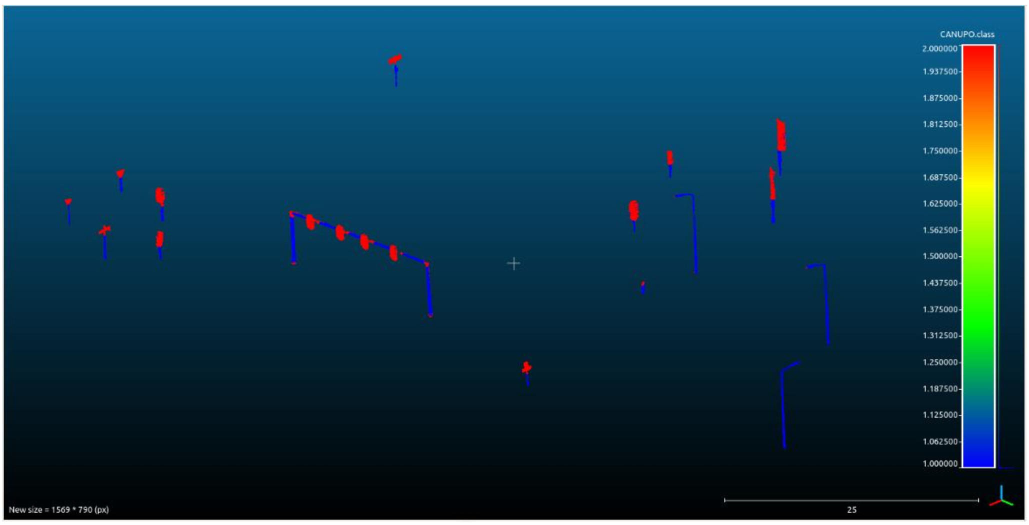


Figure 11 - Classification Result of CANUPO Algorithm : poles against all other classes

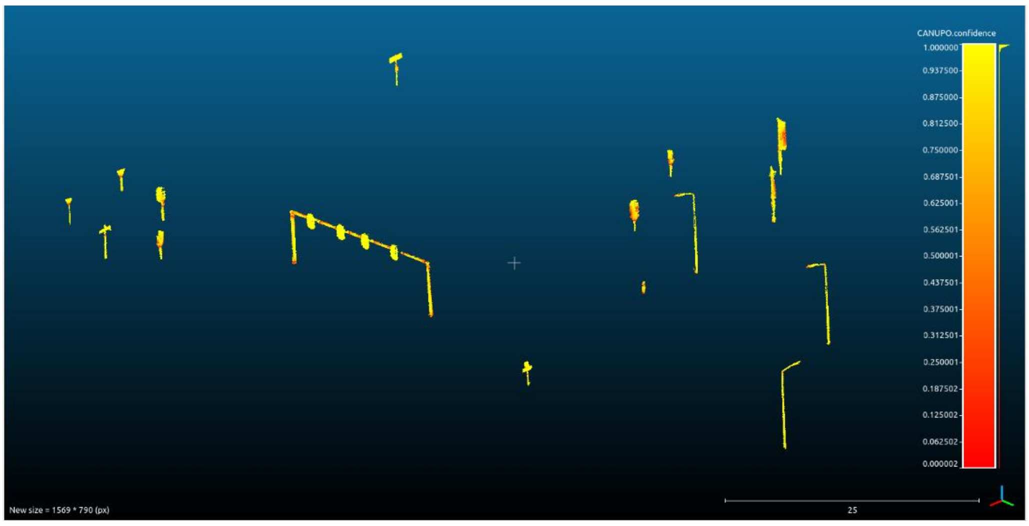


Figure 12 - The Prediction Confidence of CANUPO Algorithm

5.2. KPConv : Part-segmentation task

5.2.1. Training Step

The initial training is done solely based on the coordinate values and with default parameters set by the author. In each training step, the training accuracy and different losses are reported. By feeding the prepared data to the KPConv algorithm, without doing any further modifications, the results below would be generated:

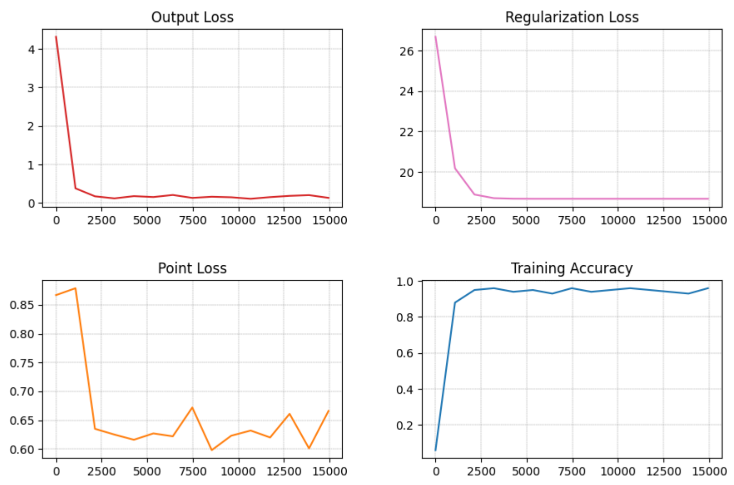


Figure 13 - Output of the training steps

Training Accuracy shows the performance of the model on the instances that it has been built upon. In other words, the model is being tested with the examples that the model has already seen. The number of correct predictions over all the predictions is considered the training accuracy.

The output loss or the segmentation loss is the mean of all the elements across all dimensions of the sparse softmax cross-entropy tensor, calculated between the true labels and logits. Logits are the probability associated with each class in the prediction for each point, calculated from the logistic regression equation.

The regularization loss is defined to increase the model's generalization capability and avoid model complexity and overfitting. In KPConv, the complexity of the model is expressed as the sum of the squares of the weights. The more the non-zero weights are, the more complex the model is, and the network suffers from overfitting.

The point loss is the sum of 'offset loss', 'gaussian loss', and the 'regularization loss'. These losses are related to the kernel extent, kernel points arrangements, and their weights. Each loss is multiplied by a predefined constant describing its importance in the point loss calculation.

As it can be seen in Figure (13), The model is converged after 2500 steps. All the outputs of the training phase remain in a same range (the training accuracy will remain above 90%, the regularization loss around 18, the output loss will be around 0.1, and the point loss is also around 0.65).

To get a better understanding of the result, we have compared it to the benchmark dataset result:

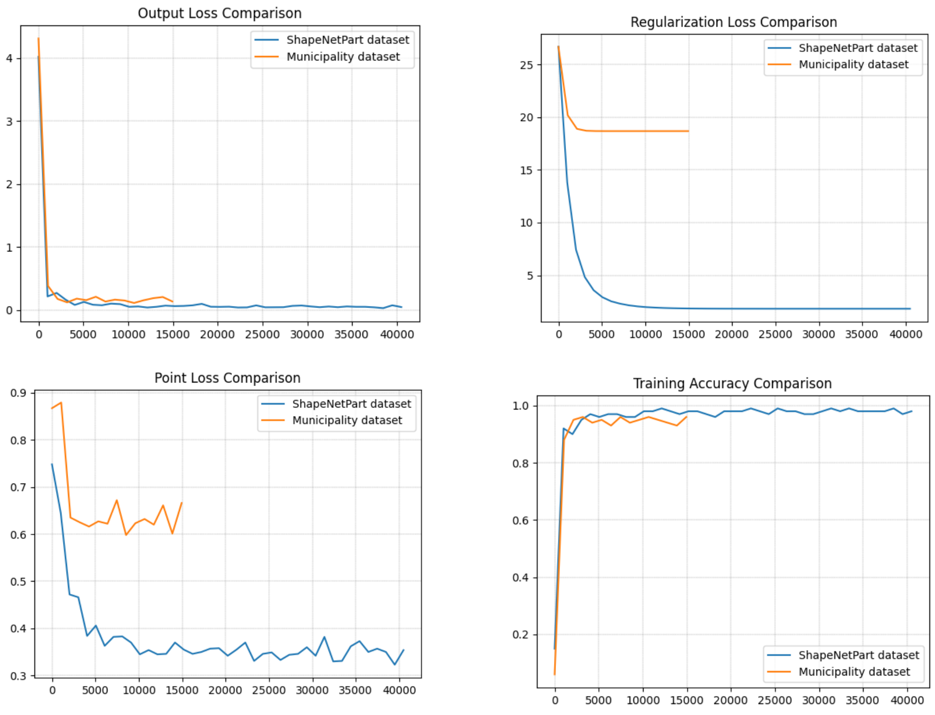


Figure 14 - Comparison of the training output on benchmark dataset (ShapeNetPart) and the custom dataset (municipality isolated poles)

As shown in figure (14), the output loss and training accuracy are comparable to the benchmark dataset result. However, a higher regularization loss can be seen for our custom dataset compared to the ShapeNetPart dataset. The higher regularization loss for our dataset shows that the non-zero weights, and thus the complexity of the model trained with our custom dataset is higher than the benchmark dataset. This might be because we expect the network to distinguish 12 classes, while in the benchmark dataset, we have a maximum of 4 classes into which the objects are being segmented.
Higher point loss for the municipality data highlights that our kernels need optimization. Maybe the inputs to the network, such as the number of kernel points, the radius of the sphere, etc., can help us get better results.
Another output from the training phase compares the results of the training set and the validation set. As shown in figure (15), The accuracy for the training set is around 88%, while for the validation set is around 80%. The trend of the lines for both sets shows that from epoch 100 onwards, the lines' pattern and their difference remain almost the same, which indicates the model not being overfitting. Also, the validation set has a bit higher output loss than the training set. Also, it has more variance, but the lines for both sets converge and have comparable values from epoch 15 onwards.

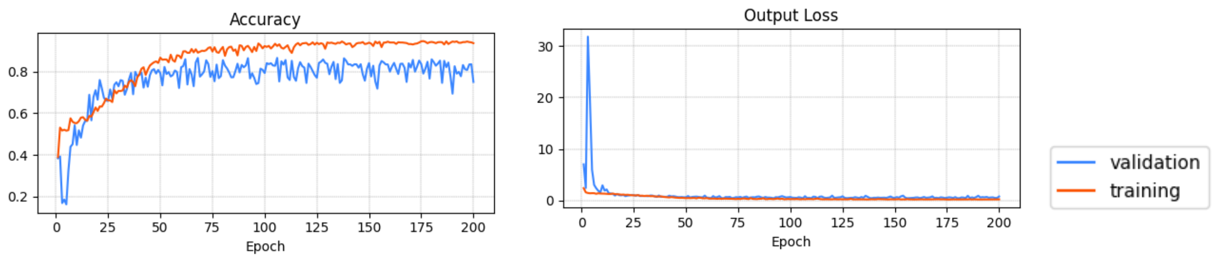


Figure 15 - Comparison of the training and validation results during the training phase

5.2.2. The Testing Results:

In the testing phase, unseen data should be given to the trained network, and the predicted label should be compared to the ground truth. The KPConv algorithm uses the coordinate values by default to part-segment 3D objects. After the part segmentation the algorithm saves ten best and ten worst predictions based on the mean intersection of union (mIoU) over all the classes.

In figure (16), three instances of the best predictions are illustrated. As it is shown, in the first instance, the upper part of the flag is confused with the pole. In the second instance, all the classes are detected for the complicated pole with the traffic sign and two traffic lights; however, the class boundaries are inaccurate and random. The existence of the same classes in both ground truth and prediction point clouds with highly inaccurate boundaries might be because of the data leakage and bugs in the code. The evaluation metrics for each class are shown in figure (17).

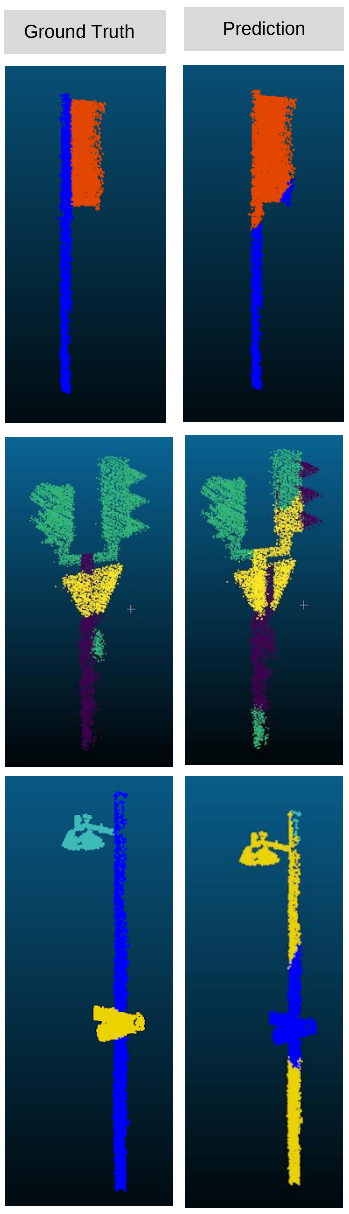


Figure 16 - Initial Results of KPConv on Municipality data | Testing Phase

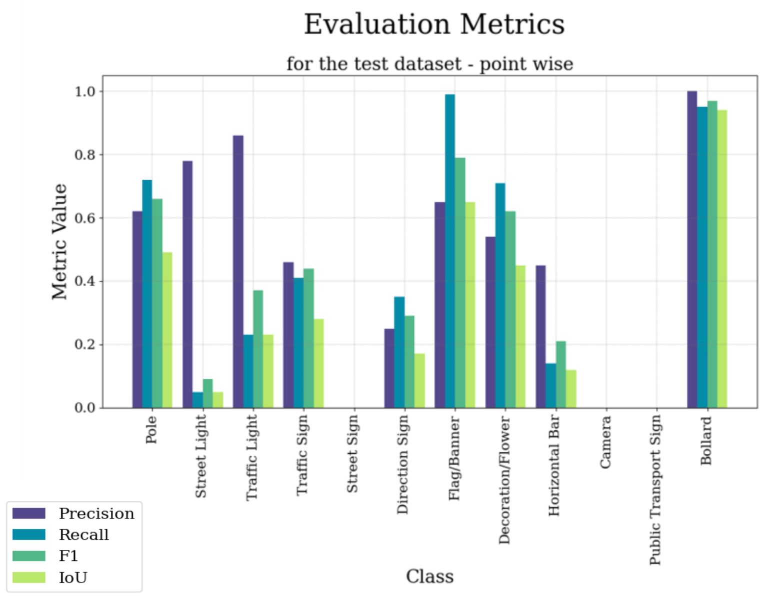


Figure 17 - Evaluation Metrics for the initial test on the KPConv algorithm

Two classes of the camera and public transport sign do not have any representative points in the test set due to the limited number of instances in the labeled dataset. There is not a single correct prediction for the street sign class. All the points predicted as street sign belong to other classes (see the third instance in Figure 16). Five classes of street light, traffic light, traffic sign, direction signs, and horizontal bars have the worst predictions. The recall value for all of these classes is less than 40% which means more than half of the points belonging to these classes are not predicted correctly.

Except for the street lights, traffic lights and horizontal bars, there are a considerable gap between IoU and Recall value for the rest of the classes. This gap confirms the inaccurate boundaries of the predicted point clouds for each attachment. This difference highlights that although the number of points predicted for class x might be close to its actual amount,

the predicted points for class x do not match the location of the ground truth points of class x.
There is also a gap between the precision and recall for classes like street light, traffic light, and horizontal bars which highlights the huge number of false negatives in the predictions.
The most significant fact to notice is the huge difference between the mIoU for the validation set reported during the training phase and the mIoU reported for the test set (Table 1). To ensure that this difference is data-partition-independent, we have done a three-fold cross-validation that led to similar results and gaps between validation and test sets.

Table 1 - mIoU results for the initial validation and test

Set	Validation	Test
mIoU	83%	36%

The reason behind the gap can be two-fold:
1) The annotated set is not big enough to train the network properly.
2) The code has some bugs or dataset-specific hard-coded lines that confuse the model and reduce its performance.

Finding an issue in the KPConv GitHub repository related to reproducing the result[8] for the ShapeNetPart dataset confirms that the low performance of the model might be related to the second reason (a bug in the code), and it is data-independent. The code fixes are discussed in the next section.

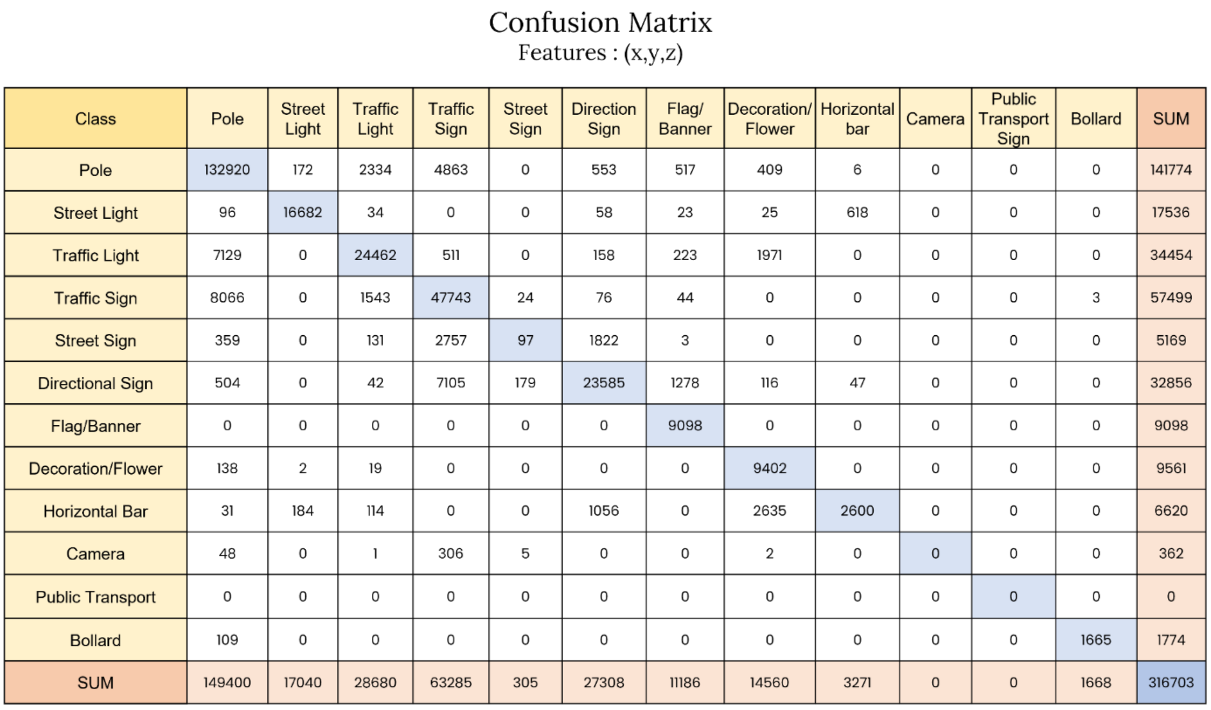
5.2.3. Code Investigation and Fix

Some unusual lines were spotted in the code, which:
1) Leak the test data to the validation set and affect the result.
2) Rotate and scale all of our instances and enclose them in a one-meter sphere. In our project with poles of different scales (from the small bollards to the medium-size poles with a traffic light or sign to the big two-sided poles), looking at all the pole instances on the same scale can be confusing for the network. The scaling makes sense for the objects that all their instances have almost the same size. Also, the rotation seems to be because of the specific coordinate system definition of the ShapeNetPart dataset.
3) Switch the y and z coordinates of the prepared test set in the testing phase, which is also reflected in one of the issues in the KPConv repository on GitHub. The y and z coordinates of the raw dataset are switched during the data preparation phase and stored for the test set, but in the test phase, while reading the prepared dataset, the code is switching the coordinates once again. The rotation in the testing phase is also confusing for the model and reduces its performance.

[8] *As a part of this project, the part-segmentation task on the ShapeNetPart dataset has been done. For this dataset the mIoU in for the validation set is 85% while for the test set a 28.5% mIoU is reported*.

By (1) making the validation set isolated from the test set, (2) reading the test prepared data without switching it in the test phase, and (3) eliminating the rotation and scaling the dataset during the training phase, the results improved significantly as follows.

Table 2 - Confusion matrix of the test result after debugging



Confusion Matrix

Features : (x,y,z)

Class	Pole	Street Light	Traffic Light	Traffic Sign	Street Sign	Direction Sign	Flag/ Banner	Decoration/ Flower	Horizontal bar	Camera	Public Transport Sign	Bollard	SUM
Pole	132920	172	2334	4863	0	553	517	409	6	0	0	0	141774
Street Light	96	16682	34	0	0	58	23	25	618	0	0	0	17536
Traffic Light	7129	0	24462	511	0	158	223	1971	0	0	0	0	34454
Traffic Sign	8066	0	1543	47743	24	76	44	0	0	0	0	3	57499
Street Sign	359	0	131	2757	97	1822	3	0	0	0	0	0	5169
Directional Sign	504	0	42	7105	179	23585	1278	116	47	0	0	0	32856
Flag/Banner	0	0	0	0	0	0	9098	0	0	0	0	0	9098
Decoration/Flower	138	2	19	0	0	0	0	9402	0	0	0	0	9561
Horizontal Bar	31	184	114	0	0	1056	0	2635	2600	0	0	0	6620
Camera	48	0	1	306	5	0	0	2	0	0	0	0	362
Public Transport	0	0	0	0	0	0	0	0	0	0	0	0	0
Bollard	109	0	0	0	0	0	0	0	0	0	0	1665	1774
SUM	149400	17040	28680	63285	305	27308	11186	14560	3271	0	0	1668	316703

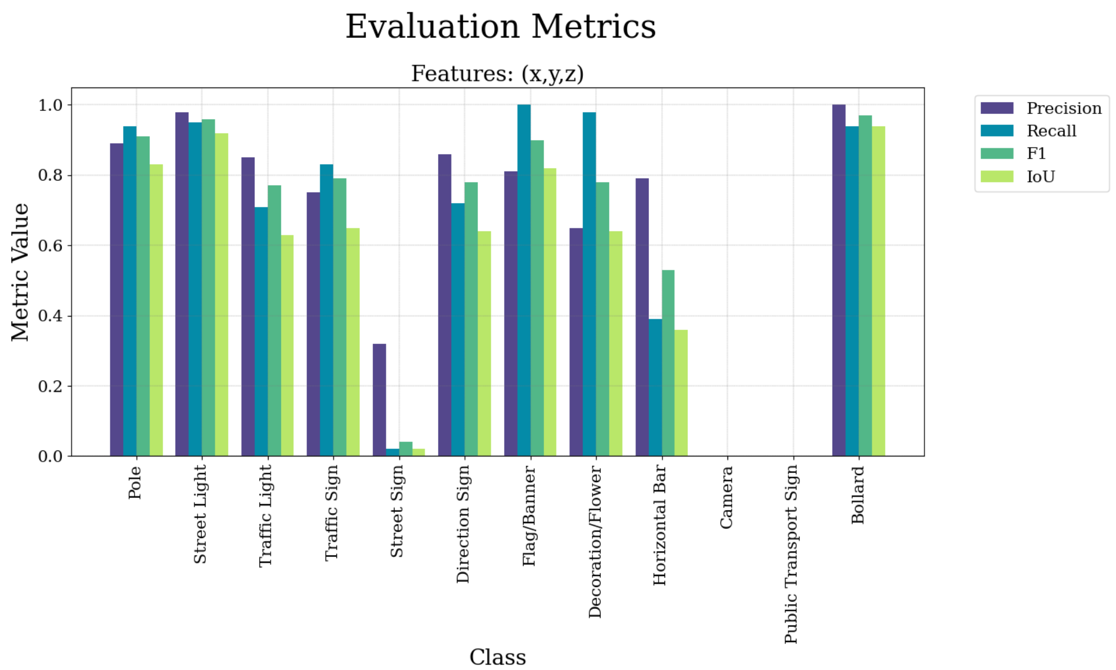


Figure 18 - Evaluation metrics after debugging

A considerable improvement can be seen in both table (2) and figure (18) among almost all the classes except street signs and bollards. The gap between the evaluation metrics for a single class is reduced for almost all the classes except street sign, direction sign, decoration and flower, and horizontal bars. The reason behind this gap can be inter-class similarity between signs (when the model is trained solely based on the coordinate values) or the limited number of training data for horizontal bars. The qualitative investigation (Figure 19) of the predicted point clouds confirms the above graph.

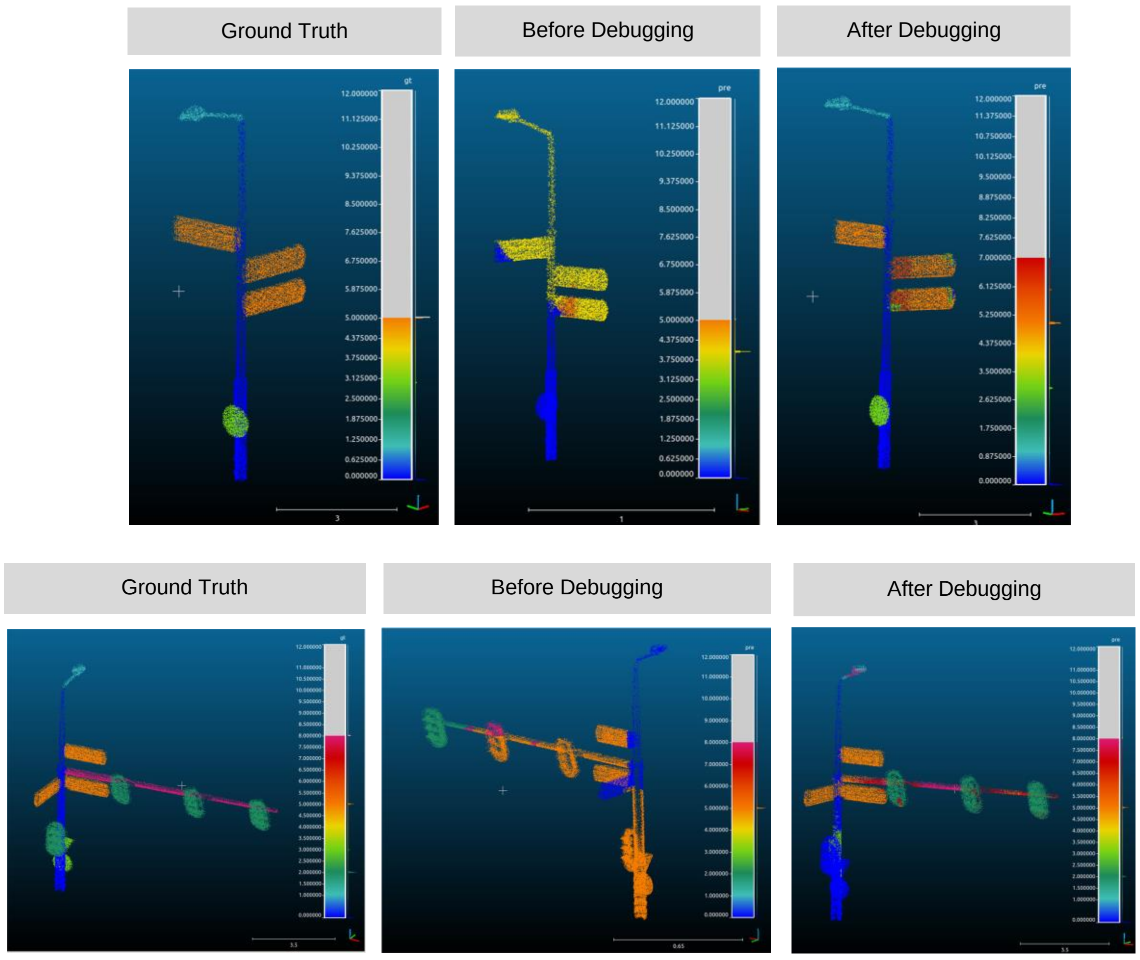


Figure 19 - Visual comparison on the test result before and after debugging

Furthermore, the test and validation mIoU gap disappears after the code modification (Table 3). The mIoU for the validation set is 87% and 84% for the test set by training the model with the default parameters, using only coordinate data as the feature. At this point, it can be claimed that the network is debugged and working correctly for decomposing pole-like objects. However, there is a possibility for improving the results (specifically required for street signs, horizontal bars), which is discussed in the following section.

Table 3 - mIoU results after code fix

Set	Validation	Test
mIoU	87.06%	84.4%

5.2.4. Improving the results

There are two strategies implemented to improve the performance of the model in this project:

1) including additional features (RGB information and intensity values)
2) Hyper-parameters fine-tuning

In the following subsections, more details are discussed about each of the strategies:

5.2.4.1. Including additional features

The KPConv algorithm uses only coordinate values to part-segment objects as default. However, the municipality dataset contains intensity and color values. This experiment aims to determine whether including additional features can improve the model's performance. Using additional information to train the network is done in 3 steps. The intensity is added as a new feature to the prepared dataset in the first step. In the second step, only RGB values and coordinates are included as the input to the model. Finally, in addition to the coordinates, both RGB and intensity values are considered the input features for the model.

The color and intensity information of the original dataset is 16-bit. Both values are first normalized[9] to feed as the extra features to the model. Normalizing the values is essential to making learning easier for the network.

1) **Including Intensity Values**

   The intensity values are affected by the angle of incidence and the distance of objects from the sensor. Therefore, calibration of the intensity values is crucial to benefit the intensity values as much as possible. However, since the sensor path is not available in the municipality data, it is assumed that the data provider has done the calibration before sharing the dataset.

   Adding intensity values benefits classes like traffic lights, traffic signs, and street signs (Figure 21). The fact that highly reflective materials are used in signs can justify the improvement in the results. However, the results get a bit worse for some classes (Figure 21), like directional signs, flags, flowers, and cameras, which can be because intensity values might be noisy and confuse the algorithm.

2) **Including RGB Values**

   Overall, adding merely RGB values is not as informative as intensity values. But still, classes like traffic lights, traffic signs, and horizontal bars are improved compared to the results from the network trained solely based on geometric information(Figure 21). Adding RGB colors can confuse the model in some cases and increase the variance of predictions in the sense that a single attachment can be predicted as multiple attachments because of the color variance or color information noises within the data (Figure 20-E).

3) **Including both Intensity and RGB**

   The most significant improvement can be spotted in the street signs, directional signs, and flower classes when including intensity and color values in the model input (Figures 20–B, 20-C and 20-D). Also, by looking at the predicted point clouds (Figure

[9] scale and mapped to be between 0 and 1

20), it is evident that the predictions have become stable, less noisy, and more accurate and balanced among all the classes (Figure 21 and Table 4).

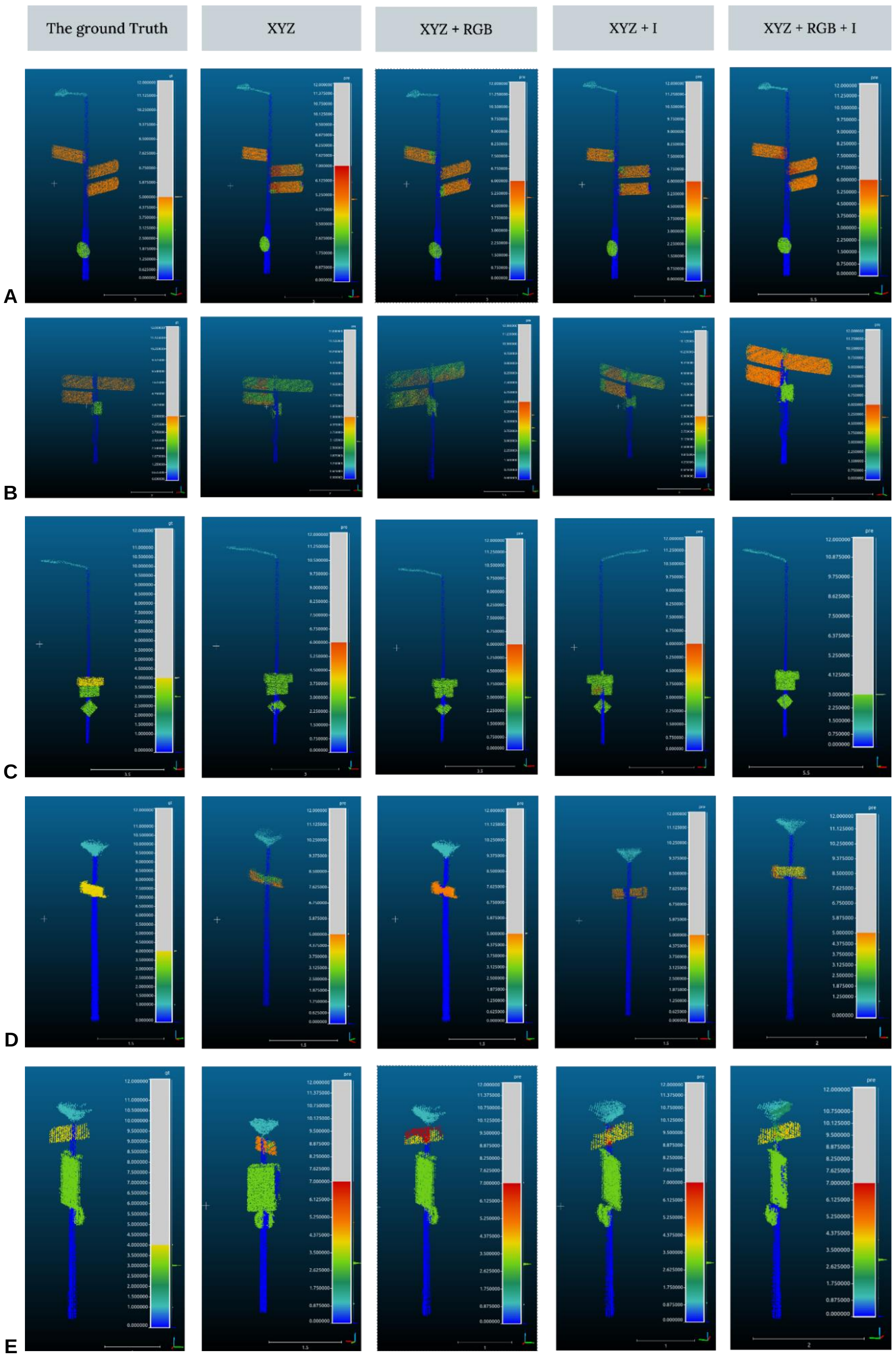


Figure 20 – A number of worst predictions by the trained networks with different combination of input features and their comparison to the ground truth

Table 4 - Comparison of mIoU on the test set for different combinations of input features

Used Features	mIoU (Test set)
XYZ	84.4%
XYZ + RGB	84.8%
XYZ + intensity	84.9%
XYZ + RGB + intensity	87.5%

An interesting observation is that adding intensity and RGB values can benefit some classes while worsening the prediction results for others (Figure 21). For instance, adding additional features is beneficial for all the classes except flags/banners and horizontal bars. For the flag/banners the best result is reached with training the network using merely coordinate values. Furthermore, The combination of RGB values and coordinates train model better to find the horizontal bars.

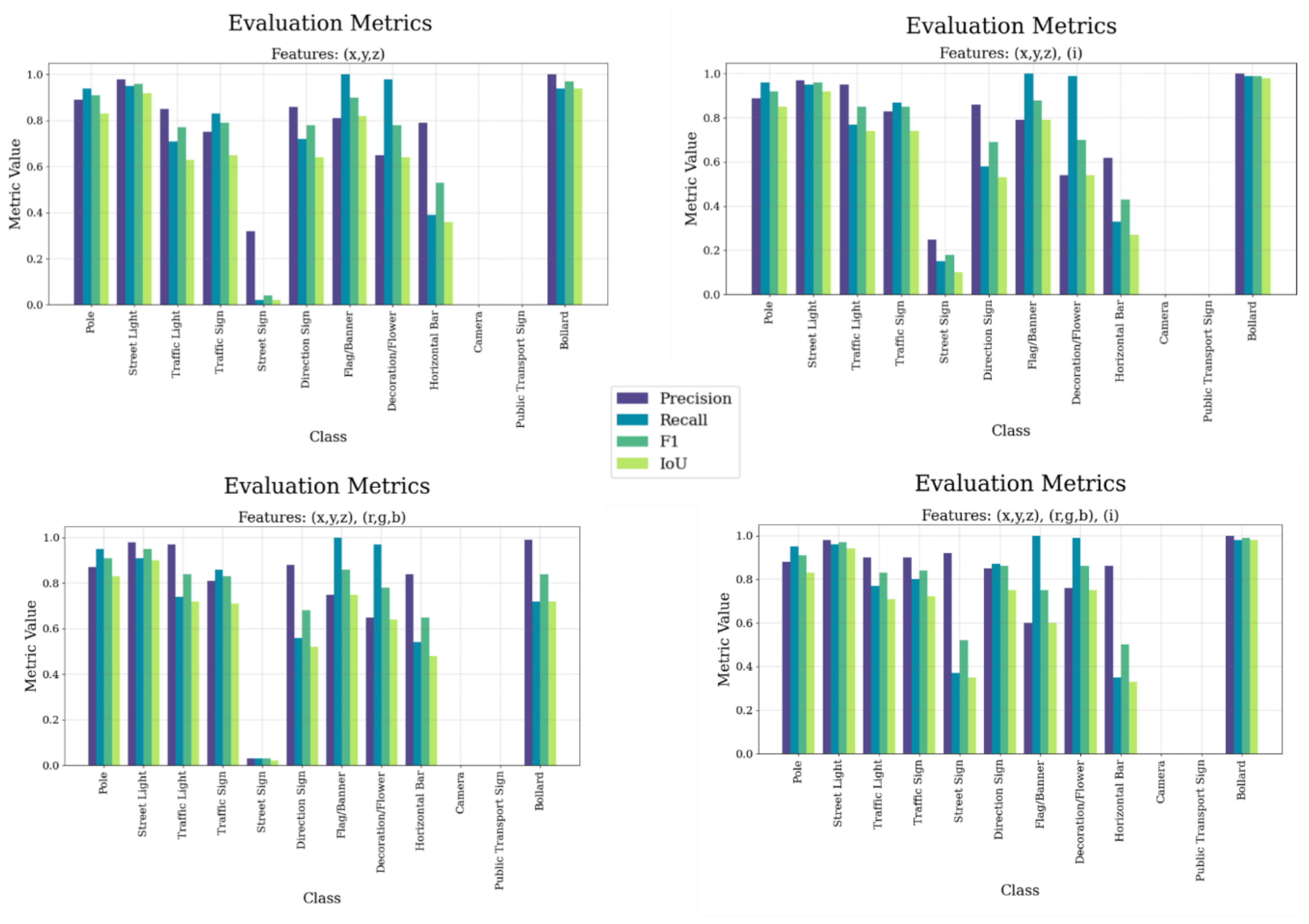


Figure 21 - Comparison of the evaluation metrics for different combinations of input features

5.2.4.2. Hyper-parameters fine-tuning

A crucial step is to set optimum values for hyperparameters as the algorithm's input to improve the prediction results. In this step, the hyper-parameter fine-tuning is done to improve the prediction for difficult instances of poles. There are four groups of input parameters for KPConv:

● **Input Parameters:** These parameters are the specifications of a task and the project. They will determine what task (semantic segmentation, part segmentation, or classification) is at hand? What object is our object of interest (an airplane, a lamp, or a pole will be decomposed)? How many parts/classes does the object have? How many features per point will be fed into the network (is it only x, y, z coordinates, or does it also contain RGB values or intensity?)What is the extent of the input clouds (rescaling the point clouds)? How many CPUs are going to be used for this task?

● **Model Parameters:** These parameters are mainly about the model architecture, the number of layers, the first feature's dimension, etc.

● **KPConv Parameters:** These are the main hyperparameters mainly describing the spherical kernel and the kernel points. The number of kernel points, the density parameter, the extent of the kernel points' influence zone, the influence mode (linear, constant, Gaussian) of the kernel points, the convolution mode (whether to use the sum or use the closest value to the point center?), and size of the grids for subsampling are determined here.

● **Training Parameters:** These parameters are mainly about the learning rate, the weights of each type of loss in calculating the total loss, the data augmentation details, the number of epochs and steps, and the number of batches and snapshot gaps.

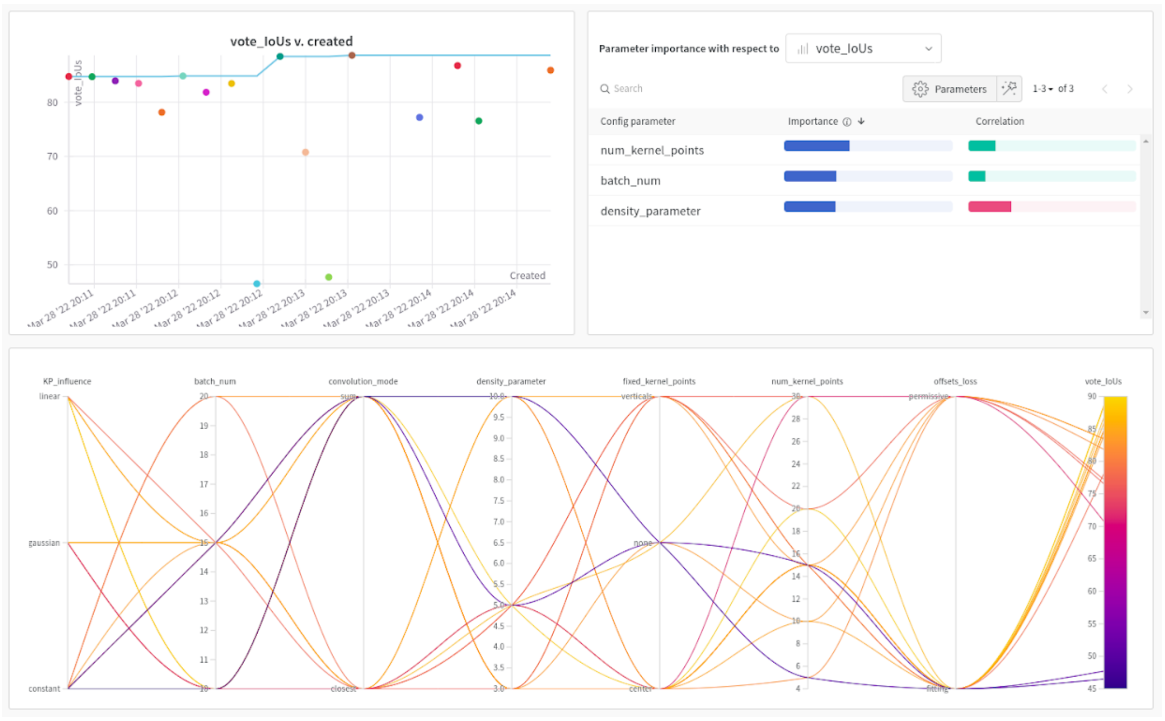
The selection of parameters for the fine-tuning phase is influenced by the dataset quality, parameter nature, parameters' role in the process, common practices, and project limitations. For instance, the Input parameters cannot be modified since they are mostly related to the task definition[10]. The only change in these parameters was including both intensity and color values as features, which resulted from the previous subsection.

The model parameters can be modified; however, they have remained intact due to the complexity of changing the model architecture and the lack of time budget.

The main parameter fine-tuning is focused on the KPConv and Training parameters to reach the optimized kernel shape, kernel points weights, and arrangement. However, a parameter like the grid size for subsampling is better to be set to the minimum value possible with respect to the point spacing to preserve the resolution of the data. Also, some parameters are better to be a multiplication of some other parameters. For example, the author claimed the Kernel Point Extent to be better set at 50 times the subsampling parameters. Therefore, some parameters are kept intact due to some reasonings, and others were selected to be changed during the fine-tuning phase.

There is a tool in the form of a website named weights and biases (wandb.ai), which can run the model with a number of different possible hyper-parameter values and find the optimized combination to maximize an evaluation metric or minimize one. For this project, the mIoU over the validation set was selected to be maximized because the validation split is being tested during the training phase and gives us a taste of the model performance on the test split. The combination of parameters is determined and altered with a random search method.

[10] *The task definition for this project is part-segmentation of poles with maximum 12 parts.*



Parameter importance with respect to vote_IoUs

Search | Parameters | 1-3 of 3

Config parameter	Importance	Correlation
num_kernel_points	🟦	🟩
batch_num	🟦	🟩
density_parameter	🟦	🟥

Figure 22 - Parameter Fine-tuning Dashboard

One of the outputs of the fine-tuning phase is the importance ranking of the parameters and their positive or negative correlation with the metric we are concerned about. As shown in figure (22), the number of kernel points is the most influential parameter. The positive correlation means that with the increase of the kernel points, the validation mIoU tends to increase; however, the correlation is not numerically significant. Also, increasing the number of batches can improve the model's performance. In contrast, the smaller density parameter helps the model learn better due to the negative correlation.

The three best combinations of parameters are shown in table (5). There is a similar pattern for selected parameters among the best combination of parameter settings, which also matches the author's default parameters. The repetitive pattern confirms that the author's default parameters are correctly selected and fine-tuned.

Table 5 - Top three combinations of parameters based on the mIoU of the validation set

parameters	KPConv Parameters					Model Parameters		Metrics	
Name	KP_ influence	Convolution_ mode	Density_ parameter	Fixed_ kernel_ point	Num_ Kernel_ points	Batch_ num	Offset_ loss	Validation mIoU	Test mIoU
Default_ parameters	linear	sum	5	center	15	15	fitting	88.45%	87.5%
Different_ sweep_12	linear	sum	6	none	10	20	fitting	88.65%	86.4%
Fearless_ sweep_24	linear	sum	6	center	30	15	fitting	82.86%	85.7%
Jolly_ sweep_14	linear	sum	3	center	20	10	fitting	88.65%	89.5%

However, investigating the prediction point clouds highlights that the quantitative result of mIoU is not descriptive enough to analyze the change in the model performance. The complexity of the pole decomposition problem leads to uncertainty about the best combination of parameters. The visual inspection of the qualitative results highlights that for a particular class of interest, there might be a particular combination of parameters that does not work the best for another class.

As shown in figure (23B), sweep 14 with more than 89% of mIoU, can better spot the traffic signs and traffic lights in a complex pole. However, it misses the horizontal bars. In other sweeps with lower mIoU, at least a part of the horizontal bar is detected, while in sweep 14, the whole horizontal pole is confused with directional signs and flags. Furthermore, the street light predictions are more stable for sweeps 12 and 24 with lower mIoU values. Figure 23A also highlights the smaller variance and higher stability for the predictions of sweep 24 with the lowest mIoU.

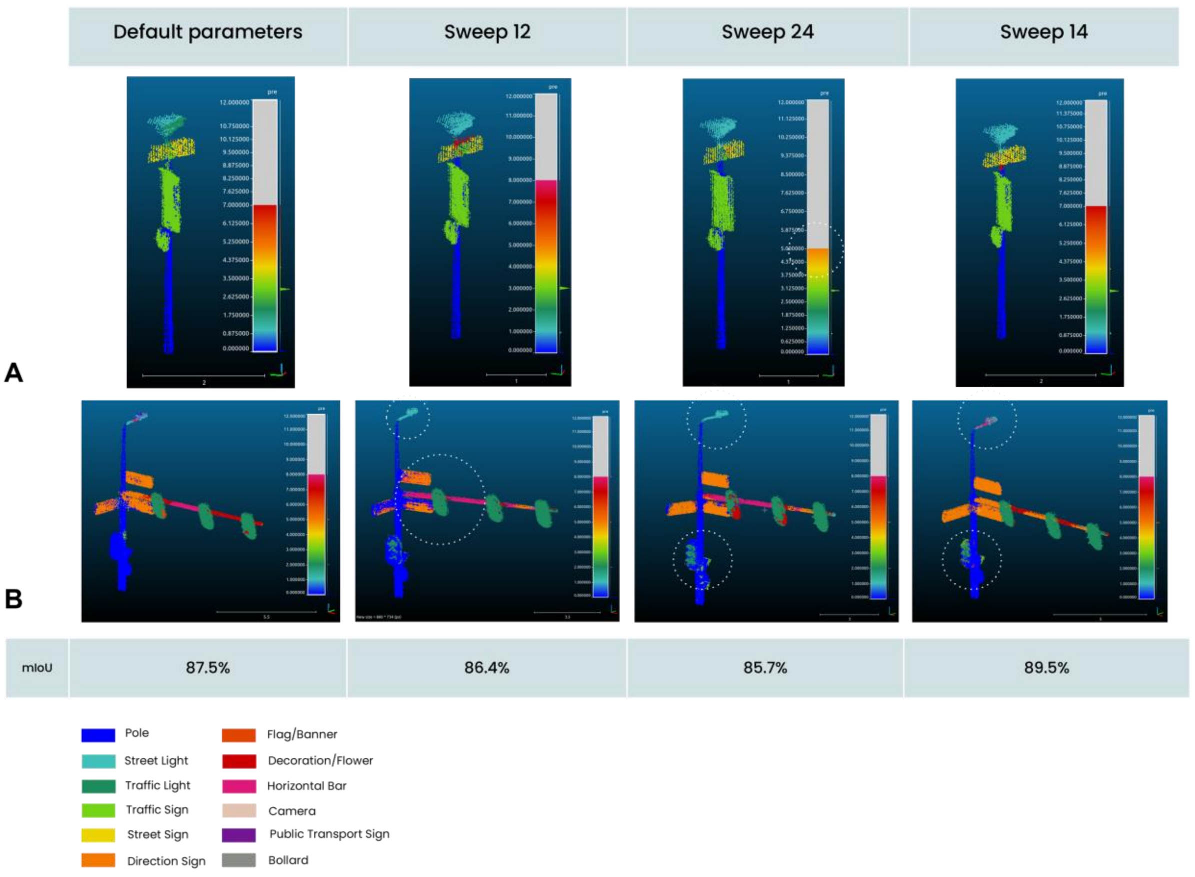


Figure 23 - Qualitative comparison of the top three prediction point clouds of the fine-tuning phase.

6. Conclusion

6.1. CANUPO Algorithm

- **Advantages:**
  - It is easy to implement this approach. It does not need any specialty to make the algorithm work.
  - It is not a black box. The whole process is transparent and understandable. It can be reproduced by a human being step by step.
  - The algorithm is developed initially to segment natural scenes. It works well in complex scenes.
  - Poles and signs can be extracted easily with acceptable accuracy, which is a good starting point for further analysis and segmentation.
  - Prediction confidence is also calculated per point, which gives a better understanding of the algorithm performance.
  - The algorithm is robust to the point density as it does not include all the points in the computation and only uses a subsampled version of the point cloud to generate the classifier.

- **Disadvantages:**
  - The input parameters require optimization based on the classes of interest, and it is hard to optimize the parameters inside the software. The optimization requires manual trial and error, which is time-consuming and cannot be validated easily.
  - Since not all objects are separable on the same scale, dealing with multiple classes requires a chain of binary classifications. In our case, it requires at least 11 classifiers. Since each classifier should be generated separately and applied separately, there is a need for a high rate of human intervention in the process. The process is not single-shot, and it is labor-intensive.
  - The order of classes in a multilabel classification task influences the result. Finding the optimum order might not also be straightforward.
  - The algorithm solely considers the local dimensionality to separate the classes. Other features like intensity, color values, or even attachment size cannot be included in the pipeline. This limitation causes inseparability among the classes with the same dimensionality, no matter how different in their shapes. Classes like traffic signs and directional signs are both planar and can be easily confused by this algorithm, while they have different plane shapes and sizes, which cannot be considered in this algorithm.
  - In this pipeline, no evaluation metrics are generated. The results can be visually investigated in the software. To have quantitative metrics, it is required to export the output and write a separate python code.

6.2. KPConv Algorithm

Advantages:

- All the attachments can be distinguished in one single pipeline and simultaneously with this method.
- The output of the algorithm can be investigated both quantitatively and qualitatively.
- The qualitative investigation of the predicted point clouds highlights that the KPConv algorithm can part-segment poles with an acceptable result with different combinations of feature inputs (coordinate values, color values, or intensity values).
- The quantitative result (the mIoU on the validation set: 84.4% and higher) confirms that the KPConv model can learn to decompose the poles based on geometric information solely or with a combination of geometric and radiometric data.
- The input features do not make a huge difference in predicting simple instances like a pole with a single attachment. However, in the case of the complex poles with multiple attachments, where the prediction of the network becomes critical, and its performance drops, the additional input features improve the predictions tangibly.
- The best numeric result of mIoU over the test set (87.5%) considering all the classes is reached when including all the possible input features (coordinates, color values, intensity values) in the training phase. In this case, the model can learn deeper and separate classes better when having additional information. After including additional features, the improvement of the model predictions is around 4% for the mIoU on the test set.
- The inference and testing time is acceptable. It approximately takes 5 minutes to generate predictions for 43 isolated pole instances with 100 votes using a 10GB GPU (GeForce RTX 2080 Ti).
- The code is flexible enough to include additional features, fine-tune hyperparameters, modify model architecture, and alter the loss functions. This robustness provides a vast opportunity for new experiments and improvements to the algorithm.

Disadvantages:

- This method requires large isolated and annotated training data. Providing this dataset is labor-intensive and time-consuming.
- The process requires time and specialty to be followed. Setting up the environment and data preparation phase might not be straightforward for beginners. Also, training the algorithm on 176 pole instances takes approximately 130 minutes with a 10GB GPU (GeForce RTX 2080 Ti).
- The deep learning process is a black box. It is not transparent enough to be followed step by step.
- The quality of predictions is highly dependent on the training set. Some of the minority classes (like cameras or public transport signs) could not be properly

learned and tested due to the limited number of instances in the annotated dataset.

- The classes of interest can affect whether to include additional features in the network. Some classes with reflective materials or distinctive colors (like street signs and direction signs) can benefit from the additional features. Meanwhile, the results for some classes with noisy intensity and colors might be affected.
- The class of interest should be considered when choosing the combination of model hyperparameters for a more stable result. While a combination of hyperparameters might generate good results for some classes, it might worsen the result for others.

6.3. Recommendations

The KPConv algorithm seems to be more promising for the municipality to use for pole decomposition because:

1) All the attachments (both for simple and complex pole instances) can be identified simultaneously with this method, with a high mIoU.
2) The municipality has already created a training set, and the code for data preparation and inference will be available as the output of this project. Therefore, an automated pipeline is already available to part-segment the poles with acceptable results.
3) The method is robust enough for further improvements, fine-tuning, or including additional features.
4) There is the opportunity to integrate the part-segmentation task with the pole-extraction task in a single pipeline as they are both point-based deep learning algorithms.

The most important aspect to consider while using the algorithm are:

1) The post-processing step is recommended to make the predictions smoothly and avoid the detection of objects not existing on a pole. For example, a limited number of points belonging to a street sign might be predicted as flags. The wrong prediction of flag points will be eliminated by smoothing the predictions and setting a threshold for the number of points representing each attachment.
2) The KPConv algorithm, with its default parameters, can part-segment poles with an acceptable mIoU. However, to achieve better results for specific classes of interest, the municipality should ensure there are enough representing points for the class of interest in the training set by adding more instances to the labeled data. Also, parameter fine-tuning and selecting features (coordinates, intensity, color values) as the inputs to the model are crucial and should be chosen considering the class of interest.

6.4. Reached milestones

The milestones I have managed to reach in my internship project are:

- Experimenting with two different algorithms, one with a machine-learning approach based on the dimensionality of the neighboring points and the other with a deep learning approach to part-segment the point cloud.

- Feeding the municipality's custom dataset to a publicly available deep learning algorithm (KPConv) and introducing a new pole class to the model.
- Debugging the KPConv codes for the part-segmentation task.
- Including additional features in the training and testing pipeline(RGB and intensity values)
- Fine-tuning a selection of parameters
- Improving the model performance.

6.5. Future works

There are a number of suggestions for improving this project that can be considered for future works, such as:

- Further fine-tuning experiments with model parameters and training parameters
- Improving the stability of the results using post-processing steps (like smoothing)
- Adding extra features like "normals" for improving the horizontal bars predictions
- Providing a larger training set with complex instances
- To make an integrated pipeline for pole extraction and decomposition

7. References

Bello, S.A., Yu, S., Wang, C., Adam, J.M., Li, J., 2020. Review: Deep learning on 3D point clouds. Remote Sensing. https://doi.org/10.3390/rs12111729

Brodu, N., Lague, D., 2012. 3D Terrestrial lidar data classification of complex natural scenes using a multiscale dimensionality criterion: applications in geomorphology.

Fukano, K., Masuda, H., 2015. Detection and classification of pole-like objects from mobile mapping data, in: ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences. Copernicus GmbH, pp. 57–64. https://doi.org/10.5194/isprsannals-II-3-W5-57-2015

Guo, Y., Wang, H., Hu, Q., Liu, H., Liu, L., Bennamoun, M., 2020. Deep Learning for 3D Point Clouds: A Survey. IEEE Transactions on Pattern Analysis and Machine Intelligence 1–1. https://doi.org/10.1109/tpami.2020.3005434

He, Y., Yu, H., Liu, X., Yang, Z., Sun, W., Wang, Y., Fu, Q., Zou, Y., Mian, A., 2021. Deep Learning based 3D Segmentation: A Survey.

Huang, J., You, S., 2015. Pole-like object detection and classification from urban point clouds, in: Proceedings - IEEE International Conference on Robotics and Automation. Institute of Electrical and Electronics Engineers Inc., pp. 3032–3038. https://doi.org/10.1109/ICRA.2015.7139615

Kang, Z., Yang, J., Zhong, R., Wu, Y., Shi, Z., Lindenbergh, R., 2018. Voxel-Based Extraction and Classification of 3-D Pole-Like Objects from Mobile LiDAR Point Cloud Data. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 11, 4287–4298. https://doi.org/10.1109/JSTARS.2018.2869801

Karlsson, R., 2014. Implementation of Advanced Monitoring Techniques in Road Asset Management-Results from the TRIMM project, Transport Research Arena.

Li, F., Elberink, S.O., Vosselman, G., 2018. Pole-like road furniture detection and decomposition in mobile laser scanning data based on spatial relations. Remote Sensing 10. https://doi.org/10.3390/rs10040531

Li, H.T., Todd, Z., Bielski, N., Carroll, F., 2021. 3D lidar point-cloud projection operator and transfer machine learning for effective road surface features detection and segmentation. Visual Computer. https://doi.org/10.1007/s00371-021-02103-8

Li, J., Cheng, X., 2022. Supervoxel-based extraction and classification of pole-like objects from MLS point cloud data. Optics and Laser Technology 146. https://doi.org/10.1016/j.optlastec.2021.107562

Li, Y., Wang, W., Li, X., Xie, L., Wang, Y., Guo, R., Xiu, W., Tang, S., 2019. Pole-like street furniture segmentation and classification in mobile LiDAR data by integrating multiple shape-descriptor constraints. Remote Sensing 11. https://doi.org/10.3390/rs11242920

Liu, R., Wang, P., Yan, Z., Lu, X., Wang, M., Yu, J., Tian, M., Ma, X., 2020. Hierarchical classification of pole-like objects in mobile laser scanning point clouds. Photogrammetric Record 35, 81–107. https://doi.org/10.1111/phor.12307

Plachetka, C., Fricke, J., Klingner, M., Fingscheidt, T., 2021. DNN-Based Recognition of Pole-Like Objects in LiDAR Point Clouds, in: 2021 IEEE International Intelligent Transportation Systems Conference (ITSC). IEEE, pp. 2889–2896. https://doi.org/10.1109/ITSC48978.2021.9564759

Pu, S., Rutzinger, M., Vosselman, G., Oude Elberink, S., 2011. Recognizing basic structures from mobile laser scanning data for road inventory studies. ISPRS Journal of Photogrammetry and Remote Sensing 66. https://doi.org/10.1016/j.isprsjprs.2011.08.006

Rodríguez-Cuenca, B., García-Cortés, S., Ordóñez, C., Alonso, M.C., 2015. Automatic detection and classification of pole-like objects in urban point cloud data using an anomaly detection algorithm. Remote Sensing 7, 12680–12703. https://doi.org/10.3390/rs71012680

Shi, Z., Kang, Z., Lin, Y., Liu, Y., Chen, W., 2018. Automatic recognition of pole-like objects from mobile laser scanning point clouds. Remote Sensing 10. https://doi.org/10.3390/rs10121891

Thanh Ha, T., Chaisomphob, T., 2020. Automated Localization and Classification of Expressway Pole-Like Road Facilities from Mobile Laser Scanning Data. Advances in Civil Engineering 2020. https://doi.org/10.1155/2020/5016783

Thomas, H., Qi, C.R., Deschaud, J.-E., Marcotegui, B., Goulette, F., Guibas, L.J., 2019. KPConv: Flexible and Deformable Convolution for Point Clouds.

Yan, L., Li, Z., Liu, H., Tan, J., Zhao, S., Chen, C., 2017. Detection and classification of pole-like road objects from mobile LiDAR data in motorway environment. Optics and Laser Technology 97, 272–283. https://doi.org/10.1016/j.optlastec.2017.06.015

Yokoyama, H., Date, H., Kanai, S., Takeda, H., 2013. Detection and Classification of Pole-like Objects from Mobile Laser Scanning Data of Urban Environments, International Journal of CAD/CAM.